Performance Testing of a Road Tolling System  
The GORT Case Study

Siegfried Goeschl  
Senior Software Engineer  
IT20ne GmbH  
siegfried.goeschl@it20one.at

Oliver Baendl  
Senior Test Analyst  
Kapsch TrafficCom  
oliver.baendl@kapsch.net

ABSTRACT
The Gauteng Open Road Tolling (GORT) system is a road tolling system covering the public highways of the Gauteng and Bakwena province in South Africa. This paper gives an overview of the GORT system and the performance testing approach being used.

Keywords
Performance Test, Stress Test, Endurance Test, Apache JMeter, WAPT Pro, soapUI Pro, Hudson, Portable Applications

1. INTRODUCTION
The GORT project is a road tolling system implemented by multiple Kapsch companies located in Austria, Argentina, Sweden and South Africa. The requirements of such a project are changing, schedules are tight, contractual penalties prohibitive, and all teams are under intense pressure to meet the next deadline. As a consequence, long-term tasks such as performance testing are moved to the very end of the schedule, as they are not immediately necessary for the next deadline. In order to minimize the risks, a dedicated performance test team was formed early on to take care of defining performance acceptance criteria, setting up a performance test environment, running performance tests regularly with the development teams, and passing the client-witnessed performance acceptance tests.

2. SYSTEM OVERVIEW
GORT is a complex system – in order to gain a better understanding, the overall system can be viewed from different perspectives such as the

- Overall system perspective
- Road-user perspective
- Functional perspective
- Performance perspective

2.1 Overall System Perspective
The overall system consists of

- 42 Road Side Stations covering 186 km of public highway
- 41 E-Toll Shops
- 11 Satellite Centers
- 20 Mobile Payment Units

- Central Operation Center (COC) houses the servers and two call-centers (TCH & VPC) with a capacity of 661 call-center agents
- Backup Operation Center (BOC) which is synchronized with COC using fiberglass cables

2.2 Road-user Perspective
A typical road-user registers his/her vehicle and payment details either using the public web-portal or by visiting the next "E-Toll Shop" or "Satellite Center" when using the public highway. During registration the road-user can choose between various tariffs and optionally purchase an electronic tag (also known as on-board unit) to be installed in the vehicle. After successful registration the road-user can happily use the public highway and the tolling fees are charged regularly. If there is a question about the recorded passages or the amount being charged, the road-user can contact a call-center to discuss any questions. In addition a registered road user with Internet access can use the self-care portal to perform common tasks such as changing registration details or viewing the recent vehicle passages.

2.3 Functional Perspective
On the public highway there are a number of tolling points (also known as road-side systems or gantries), which record vehicle passages. In order to identify the vehicle passing through a toll point, either the license plate number is determined in real-time (a so-called video passage) or the vehicle carries an electronic tag (a tag passage).

The vehicle passage data (including three vehicle images or the tag identifier) is sent to ORTBO (Open Road Tolling Back Office) over a web-service interface. At the ORTBO servers the vehicle passage data are received, verified and stored in the database.

The vehicle passages available at ORTBO are picked up by TCH (Transaction Clearing House) using stored procedures. The passages are aggregated into invoicing units and the corresponding road-user account is charged with the tolling fees. TCH also provides application servers that run the CRM back-end and the public web portal.

The VPC subsystem (Violation Processing Center) deals with the unpleasant details of running a road tolling system, such as passages of unregistered vehicles, handling the detection of blacklisted cars or mismatches between registered and actual vehicles.

The functional subsystems (e.g. ORTBO, TCH, VPC) are not only conceptual entities but also physical installations consisting of load balancers, application server farms, MSSQL cluster and/or Oracle RACs, web-service interfaces and a set of applications.
Additionally there are a few other subsystems such as OCR cluster, fingerprinting servers, Data Warehouse, a real-time traffic analysis system, backup system, system monitoring or SAN Management.

2.4 Performance Perspective
According to the requirements the overall system shall cope with the following peak loads

- 771 concurrent CRM users targeting the TCH CRM servers
- 400 concurrent web users targeting the TCH web portal
- 162 passage transactions per seconds targeting ORTBO

3. The Challenges Ahead
A lot of up-front activity is required before running the first performance tests – this chapter describes the not so obvious tasks to be accomplished.

3.1 When To Start With Performance Testing
Looking at the contractual requirements, the software and its complexity, it was obvious that it was necessary to start with performance testing activities as early as possible. It was assumed that fixing some performance bottlenecks might take long and reaching the performance target would be an iterative process. Using an iterative approach will add to the budget but minimizing delay and associated contractual penalties makes a lot more sense in financial terms.

3.2 Forming a Performance Test Team
Who would do the performance testing? The QA team was busy with manual black box testing, client acceptance tests and bug tracking, whereas the development teams were already overloaded with even more work to come. Therefore it was decided to form an independent performance testing team initially consisting of three senior software engineers. Later the team size was reduced to one part-time team member to maintain the tests and conduct the client-witnessed performance tests. Why "Senior Software Engineers" and no "Test Automation Architects" or "Test Automation Engineers"? There were no "Test Automation Architects" or "Test Automation Engineers" to be found anywhere within a reasonable time frame so the focus shifted to software developers. It was assumed that the team had to work largely independently, and a certain amount of experience would help with SQL queries, networking issues, reverse-engineering of web-services, writing test scripts, performance analysis and handling of overworked colleagues.

3.3 Defining Performance Test Scenarios
The client expects the overall system to be fast enough - so one challenge at hand is to turn client requirements and expectations into performance test and acceptance criteria.

For the GORT project the following definitions were used

Table 1. Performance Test Types

<table>
<thead>
<tr>
<th>Performance Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoke Test</td>
<td>Short test to verify the functionality</td>
</tr>
<tr>
<td>Baseline Test</td>
<td>Simulation of average load for one hour</td>
</tr>
</tbody>
</table>

For ORTBO the following numbers were derived from the performance test types above

Table 2. ORTBO Performance Test Types

<table>
<thead>
<tr>
<th>ORTBO</th>
<th>Hours</th>
<th>Passages/sec</th>
<th>Passages/Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline Test</td>
<td>1</td>
<td>40.6</td>
<td>146.000</td>
</tr>
<tr>
<td>Stress Test</td>
<td>1</td>
<td>161.9</td>
<td>583.000</td>
</tr>
<tr>
<td>Endurance Test</td>
<td>24</td>
<td>121.7</td>
<td>10.500.000</td>
</tr>
</tbody>
</table>

3.4 Agreeing on Error Rates
In a perfect world each and every performance test is error-free and no further thoughts are required. In reality, a long-running performance test causes errors when targeting a complex system. A non-exhaustive list of root causes consists of

- Spurious database deadlocks
- Dynamic table space allocations of the database
- Problems with externally hosted third-party services (e.g. payment gateway)
- Long-running garbage collection on the application server
- Latent multi-threading issues in the server code causing some "impossible" errors no one can explain or even reproduce

Consequently, an agreement on acceptable error rates and their severity is a critical success factor if client-witnessed performance tests are mandated.

3.5 Formal Performance Test Documentation
The first task at hand was writing formal performance test documents. The performance test team had a lot of experience with the technical aspects of performance testing but little experience writing formal test documentation - so where to start from here? After some research on available resources it was decided to use "Patterns & Practices Performance Testing Guidance for Web Applications" [1] as a template for the formal performance test documentation, whereas "The Top 13 Mistakes in Load Testing Applications Application" [2] makes an excellent reading to avoid common pitfalls.

3.6 Picking the Right Performance Test Tool
Based on the subsystem, their exposed interfaces and existing experience, the following tools were chosen

- soapUI Pro 3.6.x for web-service performance tests covering TCH CRM server and OEFC servers
- Apache JMeter 2.5.x for the web portal performance tests
3.7 Scheduling Performance Test Runs
A performance test has a certain impact on the system - in the best case the system performance is degraded, whereas in the worst case the performance tests breaks the system altogether, which is not very popular with the rest of the testing team. In addition, a performance test run floods the system with data to be processed, which makes certain types of functional tests more difficult. Having said that, it was agreed to schedule performance test runs after regular working hours or during weekends.

4. The Performance Test Infrastructure
What does a typical performance test infrastructure look like in many companies? A commercial performance test tool is available but only one or two installations exist in the QA department. The software developers might have no access to the performance test tool due to license restriction and/or unsupported operating systems, e.g. developing on Linux or Mac OS. Consequently, the commercial performance test tool is mostly unused by software developers, and performance testing is not part of the software development culture.

The authors strongly believe that regular performance testing needs to be part of the software development process when dealing with large distributed systems - therefore a widespread access to performance test infrastructure is essential and the following requirements shall be fulfilled

- Support of multiple operating systems - the performance tests can be run from a Windows desktop computer located in the office or from a Mac Book Pro when sitting physically in the data center
- No software installation - make it absolutely easy to move the performance test infrastructure from a QA laptop to different load injectors and back to a development team on the other side of the planet
- Everything is under version control – in our case a private Git repository on GitHub (see http://www.github.com) was used to cater for distributed development.
- High degree of automation - in order to run performance tests late at night or as part of a formal acceptance test it is mandatory to minimize manual steps and human errors.
- Command line invocation - performance tests need to be executed from the command line in order to support remote connectivity with low band-width

4.1 Portable Apps
As a starting point for the required software stack PortableApps [3] for Windows was used - PortableApps allows installing a set of applications on an USB stick and using the installed software on any Windows computer without software installation and administrator privileges. The following software was made available using Portable Apps

- Portable Free Commander as file manager
- Portable Notepad++ to stay clear of Notepad
- Portable Console2 to work with multiple command line interpreters within a single window

- Portable GIT to have access to commercial GitHub repository and access to useful Unix tools such as "less" or "grep"
- 32 & 64 Bit Java Development Kit
- Apache Ant to automate common tasks
- Hudson CI server
- Apache JMeter installation
- soapUI installation (without license file)

With such an USB stick anyone can maintain and run the performance tests without software installation on a Windows computer. It is also perfectly possible to copy the whole USB stick to the hard drive to avoid slow USB stick access. On other operating systems such as Linux or Mac OS, using environment variables to run the Java-based applications and rely on existing installations of common tools such as Bash, Midnight Commander, Git or Vim is sufficient.

4.2 Apache JMeter
Apache JMeter is an Apache project that can be used as a load testing tool for analyzing and measuring the performance of a variety of services, with a focus on web applications (see http://jmeter.apache.org).

4.3 Apache Ant
Running a repeatable performance test involves a lot of steps, which need to be automated such as

- Configuring the performance test regarding load, duration and/or test scenario
- Removing all temporary data from the last test run
- Running the performance test using JMeter or soapUI
- Sanity check of the performance test execution, e.g. running SQL scripts
- Generating HTML test reports
- Archive test reports and other artifacts
- Copying test reports and other artifacts to a FTP server

All of these steps were implemented using Apache Ant - a Java-based tool for automating software build processes. A JMeter performance test run, including all the steps above, is simply executed by running "ant clean test ftp:results" on the command line.

4.4 Hudson CI Server
Using PortableApps and Apache Ant gives you a setup-free but basic performance test infrastructure - in short you need to know your way around in order to

- Connect to the load injector box
- Locate the start directory of the performance test
- Configure the performance test scenario to be executed
- Run the tests using Ant from the command line
A better approach consists of hiding these implementation details and providing simple access over a web browser using the Hudson Continuous Integration Server (see http://hudson-ci.org) as shown below.

![Hudson CI Server](image)

**Figure 1 - Hudson CI Server**

The Hudson CI Server allows organizing the performance tests subsystems using tabs (e.g. "ER-ORT") and a list of pre-configured performance test scenarios (e.g. "er-ort-oefc-baseline") to be started on a mouse click. Additionally Hudson provides user access control, an embedded scheduler and the ability to monitor the currently executed performance test over a web browser.

5. **Lessons Learned**

Conducting extensive and/or client-witnessed performance tests results in a few lessons learned – some of them might be worth sharing.

5.1 **Achieving the Performance takes Time**

The performance testing and analysis were done in parallel with development to detect performance bottlenecks early on as suggested by [4]. The following diagram shows the measured throughput of the OEFC stress test scenario over a period of five months.

![ORTBO Stress Test Throughput](image)

**Figure 2 - ORTBO Stress Test Throughput**

It took a couple of iterations and deployments to consistently deliver the 162 request per second (below the agreed error rate), and also a lot of team effort was required - ranging from tweaking the storage, tuning the database and fixing performance issues in the application server code.

5.2 **The Perfect Performance Test Tool**

Using performance test tools seriously might reveal shortcomings and real bugs [3] - performance test tools are no exception. The following table shows the performance tools used for the subsystems under test - due to unexpected problems and/or bugs the performance test team decided to switch tools in the middle of the project.

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Delivered</th>
<th>Planned</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCH CRM Web Service</td>
<td>JMeter</td>
<td>JMeter</td>
</tr>
<tr>
<td>TCH Web Portal</td>
<td>WAPT Pro</td>
<td>JMeter</td>
</tr>
<tr>
<td>ORTBO Servers</td>
<td>JMeter</td>
<td>soapUI Pro</td>
</tr>
</tbody>
</table>

**Table 3. Performance Test Tools Usage**

5.2.1 **JMeter and ASP.NET Sites**

JMeter provides all the features to simulate web users using a browser, but testing the ASP.NET Web Portal turned out to be a challenge.

- ASP.NET server application can use advanced features such as hidden fields, JavaScript variables, parameter encryption and Ajax calls
- JMeter has no built-in knowledge about ASP.NET and its parameter wiring, which makes the script maintenance tedious and error-prone
- The web portal implementation changed regularly which broke the JMeter

In short it was impossible to fix a broken JMeter script within a few hours after a deployment. The performance test team decided to move from JMeter to WAPT Pro (see http://www.loadtestingtool.com) for web portal testing for the following reasons:

- Support for ASP.NET using a separate ASP.NET module
- Affordable license costs
- Sufficient functionality

5.2.2 **soapUI Pro**

During testing of the ORTBO web services soapUI showed multithreading issues under high load, therefore JMeter was used instead. Having said that, soapUI was still used for exploring web services and its mock server feature.

5.3 **Avoid 24 Hour Endurance Tests**

On a large system with multiple databases many activities take place after midnight such as:

- Database backup
- Creation of partitioned table spaces
- Nightly data warehouse feeds
- Execution of database maintenance scripts

These activities affect the overall system causing unexpected performance degradation and spurious system errors. If stringent acceptance criteria are applied, it is recommended either to communicate the anticipated problem, ensure that the performance tests are finished before midnight or tune the system to minimize the impact.
5.4 The Performance Tests are Broken
Whenever a critical performance issue was found, someone (usually with a job title containing "Senior" and/or "Manager") complained, "the performance test is broken" due to the following reasoning:

- The test data does not reflect reality
- The performance test script does not reflect reality
- The performance test tool is hitting its own performance bottleneck most probably caused by a central component

Unfortunately these might be completely valid points - during ORTBO testing one performance problem was indeed caused by unrealistic test data. Due to organizational safety precautions, only cars from a test fleet could be used for performance testing - this was necessary to avoid charging real road user accounts with simulated vehicle passages.

In order to guarantee that the performance test tool does not limit the measured overall throughput, the following policies were implemented:

- Using multiple and completely independent load injectors (aka "shared nothing")
- Ensuring that a half of the load injector boxes can inject twice of their required load with the same measured overall throughput

5.5 JMeter Reporting Does not Scale
JMeter reporting is available in two flavors:

- Real-time graphs available from the GUI
- HTML report created from an intermediate XML-based report file

For the GORT project the HTML report approach was used but the report creation did not scale to millions of test requests. JMeter creates an intermediate XML-based report file, which contains an entry for each executed test request. Therefore the file might be huge (exceeding 1 GByte). This XML file is transformed to HTML using XSLT, but this approach does not scale for large XML files, since most XSLT engines materialize the XML document in-memory. During the course of the project a new JMeter reporting back-end was implemented [5] which is perfectly able to process XML report files containing million of entries.

5.6 End-To-End Verification
The consultant's worst case scenario – conduct a client-witnessed performance test, report a formal success and let the management find out the next day that only half of your test request were properly processed due to a previously unknown problem in the asynchronous workflow. If that scenario becomes true your professional reputation is severely damaged and you might start looking for new career opportunities.

As a rule of thumb – whenever errors start showing up in log files or the application servers start behaving in an unexpected way assume that something went wrong and the results of the performance test run are invalid. Consequently test assertions, handcrafted SQL statements, server monitoring tools, GUI clients and DWH reports were used to ensure that all requests were indeed properly processed.

6. Conclusion
In short - performance testing is not trivial, and testing a complex system such as GORT is hard.

The performance test team was initially staffed with three software developers "to get the thing rolling" and was later scaled down to a single member. At the peak of performance testing it was a distributed, virtual and cross-functional team consisting of database administrators, lead developers, operations and QA members. Why? Writing the performance tests was the simple part – but pinpointing and fixing performance bottlenecks was hard.

The decision to rely on affordable or free performance test tools was risky – there is no big vendor to blame (remember "No one ever got fired for buying IBM"?). Nevertheless, small vendors sometimes offer limited support, mailing lists can help, and in the worst-case extra resources are required to work around issues or find a replacement. On the other hand, affordable or free test tools might find their way into the software development department permanently if no investment proposal at board level is required.

7. REFERENCES